

Lime: Data Lineage in the Malicious Environment

Michael Backes, Niklas Grimm, and Aniket Kate

Saarland University, Germany

{backes@cs, s9nigrim@stud, aniket@mmci}.uni-saarland.de

Abstract

Intentional or unintentional leakage of confidential data is undoubtedly one of the most severe security threats that organizations face in the digital era. The threat now extends to our personal lives: a plethora of personal information is available to social networks and smartphone providers and is indirectly transferred to untrustworthy third party and fourth party applications.

In this work, we present a generic data lineage framework LIME for data flow across multiple entities that take two characteristic, principal roles (i.e., owner and consumer). We define the exact security guarantees required by such a data lineage mechanism toward identification of a guilty entity, and identify the simplifying non-repudiation and honesty assumptions. We then develop and analyze a novel accountable data transfer protocol between two entities within a malicious environment by building upon oblivious transfer, robust watermarking, and signature primitives. Finally, we perform an experimental evaluation to demonstrate the practicality of our protocol.

1 Introduction

In the digital era, information leakage through unintentional exposures, or intentional sabotage by disgruntled employees and malicious external entities, present one of the most serious threats to organizations. According to an interesting chronology of data breaches maintained by the Privacy Rights Clearinghouse (PRC), in the United States alone, 867,525,654 records have been breached from 4,289 data breaches made public since 2005 [1]. It is not hard to believe that this is just the tip of the iceberg, as most cases of information leakage go unreported due to fear of loss of customer confidence or regulatory penalties: it costs companies on average \$214 per compromised record [2]. Large amounts of digital data can be copied at almost no cost and can be spread through the internet in very short time. Additionally, the risk of getting caught for data leakage is very low, as there are currently almost no accountability mechanisms. For these reasons, the problem of data leakage has reached a new dimension nowadays.

Not only companies are affected by data leakage, it is also a concern to individuals. The rise of social networks and smartphones has made the situation worse. In these environments, individuals disclose their personal information to various service providers, commonly known as *third party applications*, in return for some possibly free services. In the absence of proper regulations and accountability mechanisms, many of these applications share individuals' identifying information with dozens of advertising and Internet tracking companies.

Even with access control mechanisms, where access to sensitive data is limited, a malicious authorized user can publish sensitive data as soon as he receives it. Primitives like encryption offer protection only as long as the information of interest is encrypted, but once the recipient decrypts a message, nothing can prevent him from publishing the decrypted content. Thus it seems impossible to prevent data leakage proactively.

Privacy, consumer rights, and advocacy organizations such as PRC [7] and EPIC [8] try to address the problem of information leakages through policies and awareness. However, as seen in the following scenarios the effectiveness of policies is questionable as long as it is not possible to provably associate the guilty parties to the leakages.

Scenario 1: Social Networking. It was reported that third party applications of the widely used online social network Facebook leak sensitive private information about the users or even their friends to advertising companies [3]. In this case, it was possible to determine that several applications were leaking data by analyzing their behaviour and so these applications could be disabled by Facebook. However, it is not possible to make a particular application responsible for leakages that already happened, as many different applications had access to the private data.

Scenario 2: Outsourcing. Up to 108,000 Florida state employees were informed that their personal information has been compromised due to improper outsourcing [5]. The outsourcing company that was handed sensitive data hired a further subcontractor that hired another subcontractor in India itself. Although the offshore subcontractor is suspected, it is not possible to provably associate one of the three companies to the leakage, as each of them had access to the data and could have possibly leaked it.

We find that the above and other data leakage scenarios can be associated to an absence of accountability mechanisms during data transfers: leakers either do not focus on protection, or they intentionally expose confidential data without any concern, as they are convinced that the leaked data cannot be linked to them. In other words, when entities know that they can be held accountable for leakage of some information, they will demonstrate a better commitment towards its required protection.

In some cases, identification of the leaker is made possible by forensic techniques, but these are usually expensive and do not always generate the desired results. Therefore, we point out the need for a general accountability mechanism in data transfers. This accountability can be directly associated with *provably* detecting a transmission history of data across multiple entities starting from its origin. This is known as data provenance, data lineage or source tracing. The data provenance methodology, in the form of robust watermarking techniques [32] or adding fake data [35], has already been suggested in the literature and employed by some industries. However, most efforts have been ad-hoc in nature and there is no formal model available. Additionally, most of these approaches only allow identification of the leaker in a non-provable manner, which is not sufficient in many cases.

Our Contributions. In this paper, we formalize this problem of provably associating the guilty party to the leakages, and work on the data lineage methodologies to solve the problem of information leakage in various leakage scenarios.

As our first contribution, we define LIME, a generic data lineage framework for data flow across multiple entities in the malicious environment. We observe that entities in data flows assume one of two roles: owner or consumer. We introduce an additional role in the form of auditor, whose task is to determine a guilty party for any data leak, and define the exact properties for communication between these roles. In the process, we identify an optional non-repudiation assumption made between two owners, and an optional trust (honesty) assumption made by the auditor about the owners.

The key advantage of our model is that it enforces *accountability by design*; i.e., it drives the system designer to consider possible data leakages and the corresponding accountability constraints at the design stage. This helps to overcome the existing situation where most lineage mechanisms are applied only after a leakage has happened.

As our second contribution, we present an accountable data transfer protocol to verifiably transfer data between two entities. To deal with an untrusted sender and an untrusted receiver scenario associated with data transfer between two consumers, our protocols employ an interesting combination of the robust watermarking, oblivious transfer, and signature primitives. We also implement our protocol and demonstrate the practicality for real-life data transfer scenarios such as online social networks and outsourcing.

Paper Outline. We organize the remainder of this paper as follows: In Section 2 we introduce our model LIME and give a threat model and design goals. Section 3 describes primitives used in the paper and Section 4 presents and analyzes protocols for accountable data transfer. Section 5 shows results we obtained from a practical implementation and Section 6 gives examples of how our model can be applied to real world settings. We discuss additional features of our approach in Section 7 and related work in Section 8. Finally we present our conclusions in Section 9.

2 The Lime Framework

As we want to address a general case of data leakage in data transfer settings, we propose the simplifying model LIME (**L**ineage in the **m**alicious **e**nvironment). With LIME we assign a clearly defined role to each involved party and define the inter-relationships between these roles. This allows us to define the exact properties that our transfer protocol has to fulfill in order to allow a provable identification of the guilty party in case of data leakage.

2.1 Model

As LIME is a general model and should be applicable to all cases, we abstract the data type and call every data item *document*. There are three different roles that can be assigned to the involved parties in LIME: data *owner*, data *consumer* and *auditor*. The data owner is responsible for the management of documents and the consumer receives documents and can carry out some task using them. The auditor is not involved in the transfer of documents, he is only invoked when a leakage occurs and then performs all steps that are necessary to identify the leaker. All of the mentioned roles can have multiple instantiations when our model is applied to a concrete setting. We refer to a concrete instantiation of our model as *scenario*.

In typical scenarios the owner transfers documents to consumers. However, it is also possible that consumers pass on documents to other consumers or that owners exchange documents with each other. In the outsourcing scenario [5] the employees and their employer are owners, while the outsourcing companies are untrusted consumers.

In the following we show relations between the different entities and introduce optional trust assumptions. We only use these trust assumptions because we find that they are realistic in a real world scenario and because it allows us to have a more efficient data transfer in our framework. At the end of this section we explain how our framework can be applied without any trust assumptions.

When documents are transferred from one owner to another one, we can assume that the transfer is governed by a *non-repudiation assumption*. This means that the sending owner trusts the receiving owner to take responsibility if he should leak the document. As we consider consumers as untrusted participants in our model, a transfer involving a consumer cannot be based on a non-repudiation assumption. Therefore, whenever a document is transferred to a consumer, the sender embeds information that uniquely identifies the recipient. We call this *fingerprinting*. If the consumer leaks this document, it is possible to identify him with the help of the embedded information.

As presented, LIME relies on a technique for embedding identifiers into documents, as this provides an instrument to identify consumers that are responsible for data leakage. We require that the embedding does not affect the utility of the document. Furthermore, it should not be possible for a malicious consumer to remove the embedded information without rendering the document useless. A technique that can offer these properties is *robust watermarking*. We give a definition of watermarking and a detailed description of the desired properties in Section 3.1.

A key position in LIME is taken by the auditor. He is not involved in the transfer, but he takes action once a leakage occurs. He is invoked by an owner and provided with the leaked data. If the leaked data was

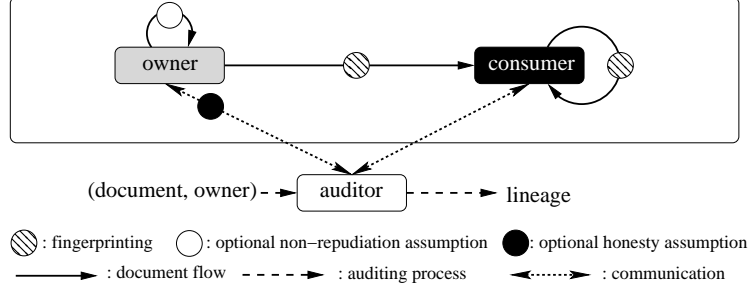


Figure 1: The LIME framework

transferred using our model, there is identifying information embedded for each consumer who received it. Using this information the auditor can create an ordered chain of consumers who received the document. We call this chain the *lineage* of the leaked document. The last consumer in the lineage is the leaker. In the process of creating the lineage each consumer can reveal new embedded information to the auditor to point to the next consumer – and to prove his own innocence. In order to create a complete lineage it is necessary that the auditor receives information from the owner, as only the owner can reveal the information embedded during the first transfer. We assume that the auditor is always invoked by the owner or that he is at least provided with information about the owners identity, so that the auditor can start his investigation with the owner and a complete lineage can be created.

We can assume that the auditor trusts the owner to be honest. Honesty in this case means, that the owner does not leak a document and blame another party. We can make this assumption as the owner is concerned with the document’s privacy in the first place. However, the auditor does not trust the consumers. In a real world setting the auditor can be any authority, for example a governmental institution, police, a legal person or even some software. In the outsourcing scenario [5], the employer can invoke the auditor who recreates the lineage and thereby uncovers the identity of the leaker. The employer can use this information to take legal actions against the outsourcing company. We show the flow of documents, the optional non-repudiation and honesty assumptions the cases in which fingerprinting is used in Figure 1. In Section 6, we present how LIME can be applied to the outsourcing scenario presented in the introduction.

Remark. We choose these honesty and non-repudiation assumptions because they realistically model many real-world scenarios. Due to these assumptions we can reduce the overhead introduced to transfers by LIME: In a transfer between two owners no fingerprinting has to be used and the owner can run a simplified transfer protocol because he is trusted by the auditor. However, these trust assumptions are not necessary for the correctness of LIME. If an owner is untrusted we can easily treat him as a consumer, so we do not have to make any trust assumptions about him.

2.2 Threat Model and Design Goals

Although we try to address the problem of data leakage, LIME cannot guarantee that data leakage does not occur in the first place; once a consumer has received a document, nothing can prevent him from publishing it. We offer a method to provably identify the guilty party once a leakage has been detected. By introducing this *reactive* accountability, we expect that leakage is going to occur less often, since the identification of the guilty party will in most cases lead to negative consequences. As our only goal is to identify guilty parties, the attacks we are concerned about are those that disable the auditor from provably identifying the guilty party.

Therefore, we consider attackers in our model as consumers that take every possible step to publish a document without being held accountable for their actions. As the owner does not trust the consumer, he

uses fingerprinting every time he passes a document to a consumer. However, we assume that the consumer tries to remove this identifying information in order to be able to publish the document safely. As already mentioned previously, consumers might transfer a document to another consumer, so we also have to consider the case of an *untrusted sender*. This is problematic because a sending consumer who embeds an identifier and sends the marked version to the receiving consumer could keep a copy of this version, publish it and so frame the receiving consumer. Another possibility to frame other consumers is to use fingerprinting on a document without even performing a transfer and publish the resulting document.

A different problem that arises with the possibility of false accusation is denial. If false accusation is possible, then every guilty receiving consumer can claim that he is innocent and was framed by the sending consumer.

The crucial phase in our model is the transfer of a document involving untrusted entities, so we clearly define which properties we require our protocol to fulfill. We call the two parties *sender* and *recipient*. We expect a transfer protocol to fulfill the following properties and only tolerate failures with negligible probabilities.

1. **correctness:** When both parties follow the protocol steps correctly and only publish their version of the document, the guilty party can be found.
2. **no framing:** The sender cannot frame recipients for the sender's leakages.
3. **no denial:** If the recipient leaks a document, he can be provably associated with it.

We also require our model to be collusion resistant, i.e. it should be able to tolerate a small number of colluding attackers [38]. We also assume that the communication links between parties are reliable.

Non-Goals. We do not aim at proactively stopping data leakage, we only provide means to provably identify the guilty party in case a leak should occur, so that further steps can be taken. We also do not aim for integrity, as at any point an entity can decide to exchange the document to be sent with another one. However, in our settings, the sender wants the receiver to have the correct document, as he expects the recipient to perform a task using the document so that he eventually obtains a meaningful result.

Our approach does not account for derived data (derived data can for example be generated by applying aggregate functions or other statistical operations), as much of the original information can be lost during the creation process of derived data. Nevertheless, we show in Section 7.1 how LIME can operate on composed data. We think of composed data as a form of data created from multiple single documents, so that the original documents can be completely extracted (e.g. concatenation of documents).

We do not consider *fairness* issues in our accountable transfer protocol; more precisely, we do not consider scenarios in which a sender starts to run the transfer protocol but aborts before a recipient received the document, or when a recipient, despite of receiving the document, falsely claims that he did not receive it. In real-world scenarios, we find fairness not to be an issue as senders and recipients expect some utility from the transfer, and are worried about their reputation and corporate liabilities.

3 Primitives

A function f is *negligible* if for all $c > 0$ there is a n_c so that for all $n \geq n_c$ $f(n) \leq \frac{1}{n^c}$. In our scheme we make use of *digital signatures*. More precisely, we use a CMA-secure signature [24], i.e., no polynomial-time adversary is able to forge a signature with non-negligible probability. For a message m that has been signed with party A 's signing key, we write $[m]_{sk_A}$. We use a symmetric encryption scheme that offers security under chosen plaintext attacks, writing $c = enc(m, ek)$ for encryption of a message m and $m = dec(c, ek)$ for decryption of a ciphertext c with symmetric key ek .

3.1 Robust Watermarking

We use the definition of watermarking by Adelsbach et al. [10]. To argue about watermarking, we need a so-called similarity function $\text{sim}(D, D')$ that returns \top if the two documents D and D' are considered similar in the used context and \perp otherwise. The similarity function is a different one for each data type used and we assume it is given.

Let \mathcal{D} be the set of all possible documents, $\mathcal{WM} \subseteq \{0, 1\}^+$ the set of all possible watermarks, \mathcal{K} the set of keys and κ the security parameter of the watermarking scheme. A *symmetric, detecting watermarking scheme* is defined by three polynomial-time algorithms:

- The probabilistic *Key Generation Algorithm* $\text{GenKey}^{\text{WM}}(1^\kappa)$ outputs a key $k \in \mathcal{K}$ for a given security parameter κ .
- The probabilistic *Embedding Algorithm* generates a watermarked document $D' = \mathcal{W}(D, w, k)$ on input of the original document $D \in \mathcal{D}$, the watermark $w \in \mathcal{WM}$ and the key $k \in \mathcal{K}$.
- The *Detection Algorithm* $\text{Detect}(D', w, D, k)$ outputs \top or \perp on input of a (potentially watermarked) document $D' \in \mathcal{D}$, a watermark $w \in \mathcal{WM}$, the original document $D \in \mathcal{D}$ and a key $k \in \mathcal{K}$. \top means that the watermark is detectable; \perp means, that it is not.

We require the following properties:

- **imperceptibility:** $\forall D \in \mathcal{D}, \forall w \in \mathcal{WM}, \forall k \in \mathcal{K}. D' \leftarrow \mathcal{W}(D, w, k) \Rightarrow \text{sim}(D, D') = \top$, i.e., the original document and the watermarked document are similar.
- **effectiveness:** $\forall D \in \mathcal{D}, \forall w \in \mathcal{WM}, \forall k \in \mathcal{K}. D' \leftarrow \mathcal{W}(D, w, k) \Rightarrow \text{Detect}(D', w, D, k) = \top$, i.e., if a watermark is embedded using a key k , the same watermark should be detectable using the same key k .
- **robustness:** For a watermarked document $D' = \mathcal{W}(D, w, k)$, $D \in \mathcal{D}, w \in \mathcal{WM}, k \in \mathcal{K}$ there is no polynomial-time adversary that can compute a $D'' \in \mathcal{D}$ given D' and w so that $\text{sim}(D', D'') = \top$ and $\text{Detect}(D'', w, D, k) = \perp$ with non-negligible probability. This means that no adversary can efficiently remove or change a watermark without rendering the document unusable.

Additionally, we require our watermarking scheme to support *multiple re-watermarking*, i.e., it should allow for multiple (bounded by the dataflow path length) watermarks to be embedded successively without influencing their individual detectability. This property can also be considered as a special kind of robustness, as it prevents adversaries from making a watermark undetectable simply by adding more watermarks using the same algorithm. More information and some experimental results about this property can be found in [32].

We also expect the watermarking scheme to be *collusion resistant* [30], i.e., even if an attacker can obtain differently watermarked versions of a document, he should not be able to create a version of the document where none of these watermarks is detectable. Further, for some watermarking schemes the input of the original document is not required for detection of watermarks. We call those watermarking schemes *blind*. As already stated in [10] this definition of watermarking is very strong and its properties are not yet provided by available schemes. Although we chose this strong definition to prove the correctness of our scheme, there are existing schemes whose properties are arguably sufficient in practice such as the Cox watermarking scheme [17].

Cox Watermarking Scheme [17]. This scheme uses spread-spectrum embedding of Gaussian noise to watermark images. To provide robustness, the watermark is embedded in the most significant part of the picture, so that removing the watermark should not be possible without destroying the underlying picture. The α -factor of the algorithm is a parameter that determines how strong the Gaussian noise is influencing the original image, thus it can be used to trade robustness against imperceptibility. Although our strong robustness requirement is not formally fulfilled, it is shown that the scheme is robust against many common attacks such as scaling, JPEG compression, printing, xeroxing and scanning, multiple re-watermarking and others [17].

Initialization		
Sender		Chooser
Pick random $C, r \in Z_q$ and compute g^r and C^r . Send C .	\Rightarrow	
Transfer		
Sender holding M_0, M_1		Chooser requesting M_σ
		Pick random $1 \leq k \leq q$ and compute $PK_\sigma = g^k$ and $PK_{1-\sigma} = C/PK_\sigma$. \Leftarrow Send PK_0 .
Compute PK_0^r and $PK_1^r = C^r/PK_0^r$. Encrypt M_0 as $E_0 = H(PK_0^r) \oplus M_0$ and M_1 as $E_1 = H(PK_1^r) \oplus M_1$. Send g^r, E_0, E_1 .	\Rightarrow	Compute $H((g^r)^k) = H(PK_\sigma^r)$ and use it to decrypt M_σ .

Figure 2: Oblivious Transfer protocol by Naor and Pinkas [34]

3.2 1-out-of-2 Oblivious Transfer

1-out-of-2 Oblivious Transfer (OT_1^2) involves two parties, the *sender* and the *chooser*. The sender offers two items M_0 and M_1 and the chooser chooses a bit σ . The chooser obtains M_σ but no information about $M_{1-\sigma}$ and the sender learns nothing regarding σ . In this context, when speaking of learning nothing, we actually mean nothing can be learned with non-negligible probability. There are different concrete instantiations of this primitive.

As an example implementation of OT_1^2 we show a protocol by Naor and Pinkas [34]: The protocol operates over a group Z_q of prime order q with a generator g for which the computational Diffie-Hellman assumption holds. The protocol is proven secure in the random oracle model using an ideal hash function H . The construction can be found in Figure 2.

In the initialization phase, the sender picks a random group element C and sends it to the chooser. It is important that the chooser does not know $DLog(C)$. Additionally, the sender chooses another random group element r and computes C^r . These values are not used in the initialization phase, but for efficiency reasons the sender can compute these values in this phase, as it can be performed offline.

In the transfer phase, the chooser picks a random $1 \leq k \leq q$ and computes $PK_\sigma = g^k$ and $PK_{1-\sigma} = C/PK_\sigma$, so that he knows $DLog(PK_\sigma)$. Then he sends PK_0 to the sender. The sender computes PK_0^r and without further exponentiation $PK_1^r = C^r/PK_0^r$. Then he encrypts the two messages M_0 and M_1 as $E_0 = H(PK_0^r) \oplus M_0$ and $E_1 = H(PK_1^r) \oplus M_1$ and sends g^r, E_0 and E_1 to the chooser. The chooser can now compute $H((g^r)^k) = H(PK_\sigma^r)$ and so decrypt $M_\sigma = E_\sigma \oplus H(PK_\sigma^r)$.

The proof for the correctness and security of the protocol in the random oracle model and under the Diffie-Hellman assumption can be found in [34]. Additionally, they provide an approach to improve efficiency for multiple executions of OT_1^2 using a bandwidth/computation tradeoff. This is achieved by performing one single instance of $OT_1^{2^n}$ offering all 2^n possible combinations instead of n instances of OT_1^2 offering every part on its own. Depending on available computational power and bandwidth, this can improve efficiency.

When we use OT_1^2 in our protocols to send messages, the sender actually encrypts the messages, sends both encryptions to the chooser and performs OT_1^2 just on the decryption keys. This allows us to use the OT_1^2 protocol with a fixed message size while actually sending messages of arbitrary size. Note that this

sender holding D	recipient requesting D_w
$k \leftarrow \text{GenKey}^{WM}(1^\kappa)$	
$\sigma = (C_S, C_R, \tau)$	
$D_w = \mathcal{W}(D, \sigma, k)$	$\implies \Downarrow D_w$

Figure 3: protocol for trusted senders

could only be a security risk if the chooser was able to break the encryption scheme.

4 Accountable Data Transfer

In this section we specify how one party transfers a document to another one, what information is embedded and which steps the auditor performs to find the guilty party in case of data leakage. We assume a public key infrastructure to be present, i.e. both parties know each others signature verification key.

4.1 Trusted Sender

In the case of a trusted sender it is sufficient for the sender to embed identifying information, so that the guilty party can be found. As the sender is trusted, there is no need for further security mechanisms. In Figure 3, we present a transfer protocol that fulfills the properties of correctness and no denial as defined in Section 2.2. As the sender is trusted to be honest, we do not need the no framing property.

The sender, who is in possession of some document D , creates a watermarking key k , embeds a triple $\sigma = (C_S, C_R, \tau)$ consisting of the two parties' identifiers and a timestamp τ into D to create $D_w = \mathcal{W}(D, \sigma, k)$. He then sends D_w to the recipient, who will be held accountable for this version of the document. As the sender also knows D_w , this very simple protocol is only applicable if the sender is completely trusted; otherwise the sender could publish D_w and blame the recipient.

4.2 Untrusted Sender

In the case of an untrusted sender we have to take additional actions to prevent the sender from cheating, i.e. we have to fulfill the no framing property. To achieve this property, the sender divides the original document into n parts and for each part he creates two differently watermarked versions. He then transfers one of each of these two versions to the recipient via OT_1^2 . The recipient is held accountable only for the document with the parts that he received, but the sender does not know which versions that are. The probability for the sender to cheat is therefore $\frac{1}{2^n}$. We show the protocol in Figure 4 and provide an analysis of the protocol properties in Section 4.4.

First, the sender generates two watermarking keys k_1 and k_2 . It is in his own interest that these keys are *fresh* and *distinct*. The identifying information that the sender embeds into the document D is a signed statement $\sigma = [C_S, C_R, \tau]_{sk_{C_R}}$ containing the sender's and recipient's identifiers and a timestamp τ , so that every valid watermark is authorized by the recipient. The sender computes the watermarked document $D' = \mathcal{W}(D, \sigma, k_1)$, splits the document D' into n parts and creates two different versions $D_{i,j} = \mathcal{W}(D_i, j, k_2)$ of each part by adding an additional watermark $j \in \{0, 1\}$. For each version of each part $D_{i,j}$, $i \in \{1, \dots, n\}$, $j \in \{0, 1\}$ he creates a signed message $m_{i,j} = [\tau, i, j]_{sk_{C_S}}$ containing the timestamp of the current transfer, the part's index and the content of the version's second watermark. Then he generates an AES key $ek_{i,j}$, encrypts $c_{i,j} = \text{enc}(\langle D_{i,j}, m_{i,j} \rangle, ek_{i,j})$ and sends $c_{i,j}$ to the recipient. The recipient chooses a bit $b_i \in \{0, 1\}$ for each $i \in \{1 \dots n\}$ and receives ek_{i,b_i} via oblivious transfer (note that the n executions of OT_2^1 can be parallelized). He then decrypts all c_{i,b_i} using ek_{i,b_i} and reconstructs the document by joining the parts $D_{1,b_1} \dots D_{n,b_n}$. The signed statements $m_{1,b_1} \dots m_{n,b_n}$ serve as proof of his

choice. As for each part he chooses a bit b_i , the final version is identified by a bitstring $\bar{b} \in \{0, 1\}^n$. As he will only be held accountable for the version watermarked with \bar{b} , we have a failure probability (i.e., the sender correctly guessing \bar{b}) of $\frac{1}{2^n}$.

4.3 Data Lineage Generation

The auditor is the entity that is used to find the guilty party in case of a leakage. He is invoked by the owner of the document and is provided with the leaked document. In order to find the guilty party, the auditor proceeds in the following way:

1. The auditor initially takes the owner as the current suspect.
2. The auditor appends the current suspect to the lineage.
3. The auditor sends the leaked document to the current suspect and asks him to provide the detection keys k_1 and k_2 for the watermarks in this document as well as the watermark σ . If a non-blind watermarking scheme is used, the auditor additionally requests the unmarked version of the document.
4. If, with key k_1 , σ cannot be detected, the auditor continues with 9.
5. If the current suspect is trusted, the auditor checks that σ is of the form (C_S, C_R, τ) where C_S is the identifier of the current suspect, takes C_R as current suspect and continues with 2.
6. The auditor verifies that σ is of the form $[C_S, C_R, \tau]_{sk_{C_R}}$ where C_S is the identifier of the current suspect. He also verifies the validity of the signature.
7. The auditor splits the document into n parts and for each part he tries to detect 0 and 1 with key k_2 . If none of these or both of these are detectable, he continues with 9. Otherwise he sets b'_i as the detected bit for the i th part. He sets $\bar{b}' = b'_1 \dots b'_n$.
8. The auditor asks C_R to prove his choice of $\bar{b} = b_1 \dots b_n$ for the given timestamp τ by presenting the $m_{i,b_i} = [\tau, i, b_i]_{sk_{C_S}}$. If C_R is not able to give a correct proof (i.e., m_{i,b_i} is of the wrong form or the signature is invalid) or if $\bar{b} = \bar{b}'$, then the auditor takes C_R as current suspect and continues with 2.
9. The auditor outputs the lineage. The last entry is responsible for the leakage.

Remark. It would also be possible to include the signed statement σ in every single part of the document, but as the maximum size of a watermark is limited by the document's size, this might be problematic for scenarios where the parts are small. Therefore, we embed σ in the complete original document and only embed single bits to the (possibly small) parts of the document. We use a timestamp τ to uniquely identify a specific transfer between two parties, and thus assume that no two transfers between the same two parties take place at the same time. However, it would be possible to use a counter that is incremented on each transfer to allow multiple transfers at the exact same time.

4.4 Analysis of the Protocol

We now show that the protocol presented in Figure 4 fulfills the required properties of correctness, no framing and no denial as presented in Section 2.2 :

1. **correctness:** Assume that both parties follow the protocol steps correctly. We show that for all possible scenarios the guilty party is determined correctly:

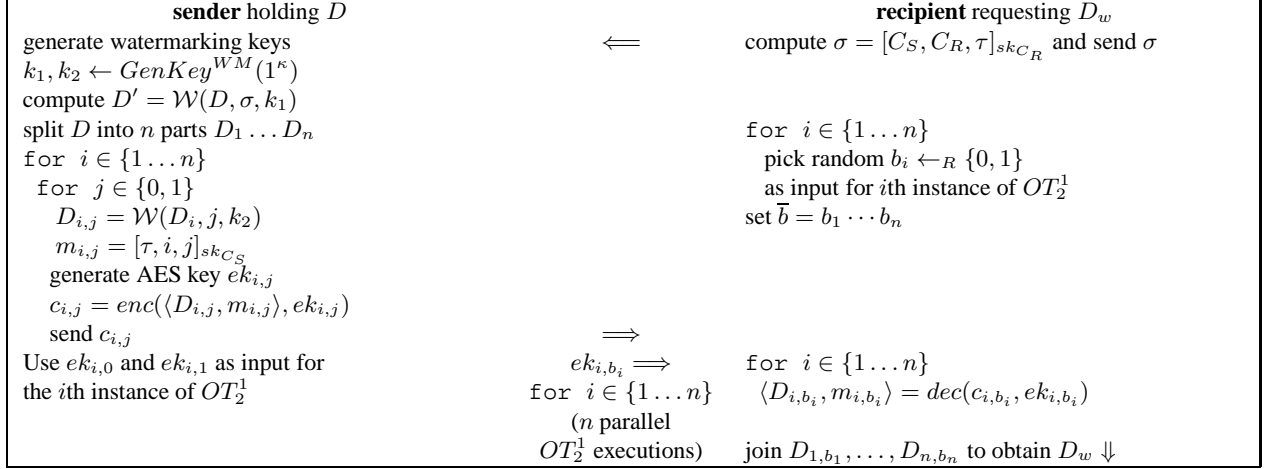


Figure 4: protocol for untrusted senders

- (a) *the sender publishes D or D'* : The auditor does not detect σ (in the case of D) or the b_i (in the case of D') and correctly blames the sender.
- (b) *the recipient publishes D_w* : The auditor successfully detects σ and \bar{b}' and verifies that σ is of the correct form. The recipient is able to provide the proof of his choice of \bar{b} ; the auditor verifies $\bar{b}' = \bar{b}$ and suspects the recipient. As there are no further watermarks embedded, the auditor correctly blames the recipient.

2. **no framing**: In a first step we show that the sender cannot obtain the version of the document for which the recipient can prove his choice (i.e. the version watermarked with the bitstring \bar{b}): The sender knows all the $D_{i,j}$, that are used for the computation of D_w , but he does not know the bitstring \bar{b} that the recipient chose due to the properties of oblivious transfer. So all he can do is guess a bitstring $\bar{b}^* = b_1^* \dots b_n^*$ and create $D_w^* = \text{join}(D_{1,b_1^*}, \dots, D_{n,b_n^*})$. But \bar{b} and \bar{b}^* (and therefore D_w and D_w^*) are the same only with negligible probability of $\frac{1}{2^n}$, so the probability for the sender to learn D_w is negligible if he follows the protocol correctly.

The sender might try to learn about \bar{b} by offering the same version twice during the oblivious transfer. Usually the recipient would have no possibility of realizing this, as he cannot detect the watermark, but as the sender additionally has to send the signed statements $m_{i,j} = [\tau, i, j]_{sk_{C_S}}$ the recipient knows that he received what he asked for. This is the case because j has by construction the same value as b_i chosen by the recipient. The sender can still send a wrong version $D_{i,1-j}$ and so know the bitstring the document is watermarked with, but as the recipient only proves his *choice* of b , the sender still cannot frame him; he would only lose the ability to prove the recipients' guilt in case the recipient publishes the document.

In a second step we show that a malicious sender cannot create a document that the recipient will be held accountable for without running the transfer protocol:

As the correctness of the signed statement σ is verified in the auditing process and as the sender can only forge the recipient's signature with negligible probability, the only possibility to mount this attack is to reuse a valid signed statement from a past transaction. This implies that the included timestamp τ is the same, too. As the auditor asks the recipient to prove his choice of \bar{b} for this τ , the recipient is able to provide a correct proof, as a valid transfer with timestamp τ actually happened. Analogous to the previous case, the sender can only chose $\bar{b}^* \in \{0, 1\}^n$ randomly and therefore he can only succeed with negligible probability.

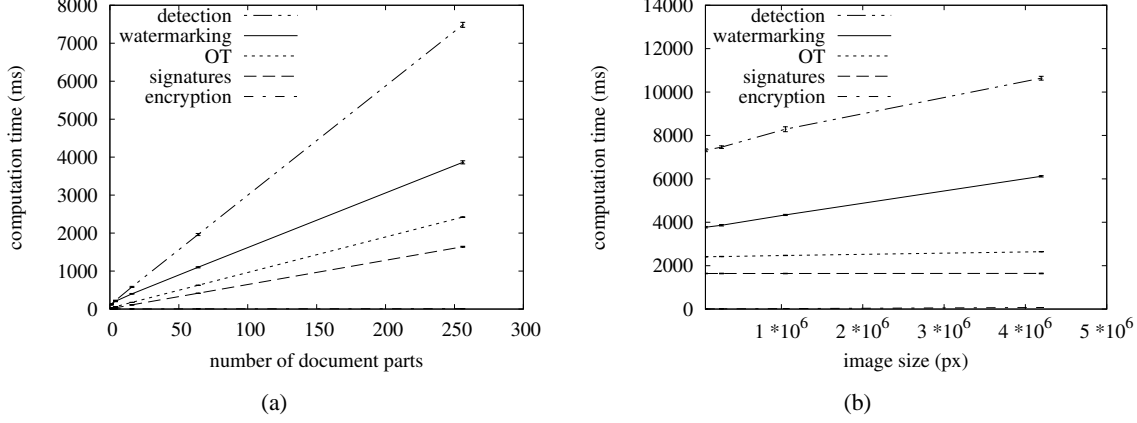


Figure 5: (a) shows computation times for different numbers of document parts; (b) shows computation times for different image sizes.

From these two steps it follows that the sender is not able to frame a recipient.

3. **no denial:** We first show that the recipient cannot publish a version of the document whose embedded watermarks are different from those embedded in the version D_w of the document that he righteously obtained. He also cannot obtain the watermark-free version D : As the recipient only receives the watermarked version, he could only learn D by removing the watermark, which he can only do with negligible probability due to the robustness property of the watermarking scheme. The recipient could also create a watermarked version with a different bitstring embedded, if he is able to get $D_{i,1-b_i}$ for some i , but this is only possible if he breaks the OT_2^1 scheme or the encryption scheme, which is only possible with negligible probability.

We now show that a recipient cannot cheat during the auditing process, when he proves which version of the document he asked for during the transfer protocol: In order to prove another choice of \bar{b} he would again have to break the OT_2^1 scheme or the encryption scheme in order to learn the $m_{i,1-b_i}$ for some i or he would need to forge the sender's signature to create $m_{i,1-b_i}$ for some i . As all of this is only possible with negligible probability, the overall probability for the recipient to succeed is negligible.

From these two steps it follows that the recipient is not able to publish a document without being caught.

This proves our protocol's security up to fairness issues. However, as discussed in Section 2.2, we do not find fairness issues to be problematic in our application scenarios.

5 Implementation and Microbenchmarking

We implemented the protocol in Figure 4 as a proof-of-concept and to analyze its performance. For the oblivious transfer subprotocol we implemented the protocol by Naor and Pinkas [34] using the PBC library [6], which itself makes use of the GMP library [4]. For signatures we implemented the BLS scheme [14], also using the PBC library. For symmetric encryption we used an implementation of AES from the Crypto++ [19] library. For watermarking we used an implementation of the Cox algorithm for robust image watermarking [17] from Peter Meerwald's watermarking toolbox [33]. We set the α -factor, which determines the strength of the watermark, to a value of 0.1.



Figure 6: (a) shows an image transferred with our algorithm. In (b) we set the α -factor used by the Cox algorithm to 0.5 in order to obtain very strong watermarks for the small parts. As a result differences between adjacent document parts are visible.

We executed the experiment with different parameters to analyze the performance. The sender and recipient part of the protocol are both executed in the same program, i.e., we do not analyze network sending, but only computational performance. The executing machine is a Lenovo ThinkPad model T430 with 8 GB RAM and $4 \times 2.6\text{GHz}$ cores, but all executions were performed sequentially. We measured execution times for different phases of the protocol: watermarking, signature creation, encryption, oblivious transfer and detection. We executed each experiment 250 times and determined the average computation time and the standard deviation.

In the first experiment we used an image of size 512×512 pixels and changed the number of parts the image was split into. We show the results in Figure 5(a). We can see that the execution time of watermarking, signatures, oblivious transfer and detection is linear in the number of document parts. The execution time of encryption is also increasing slowly, but it is still insignificant compared to the other phases.

In the second experiment we used a fixed number of document parts of 256 and changed the size of the image used. We used 4 different sizes: 256×256 px (65KB), 512×512 px (257KB), 1024×1024 px (1.1 MB) and 2048×2048 px (4.1 MB). The results can be seen in Figure 5(b). We can observe that the execution time for watermarking and detection is linear in the image size, but in contrast to the previous result, we already start with a long execution time for small sizes and the growth is rather slow. The execution time for oblivious transfer stays constant, which might be surprising at first sight, but this can be explained easily, as we only transfer AES keys in the oblivious transfer phase and these are of constant size for all image sizes. Of course the encrypted files also have to be sent over the network (so there would be a higher communication overhead for bigger images), but in our implementation we only considered the computational costs. The execution time for the creation of the signatures is also constant as the number and form of the signed statements is the same for all images. Again the execution time of encryption is increasing slowly, but it is of no significance compared to the other phases.

The tests showed that from the resulting files all watermarks (i.e. the identifying watermark σ and all the

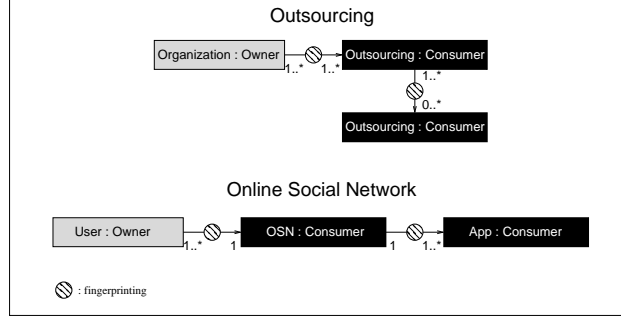


Figure 7: Outsourcing Scenario

single bits b_i forming the bitstring \bar{b}) could be correctly detected, showing the correctness of our protocol. In Figure 6(a) in the appendix we give an example of an image that was transferred using our algorithm.

We find that latencies of a few seconds are acceptable in the scenarios that we considered. Additionally, as we show in our experiments in Figure 5(a), one can easily perform a tradeoff between performance and security. It is also possible to use the OT extension technique presented in [27] to increase the efficiency of oblivious transfer.

Although we use only image files as documents in our experimental implementation, we stress that the same mechanism can be used for all types of data for which robust watermarking schemes exist.

6 Scenarios

6.1 Outsourcing

The first diagram in Figure 7 shows a typical outsourcing scenario. An organization acts as owner and can outsource tasks to outsourcing companies which act as consumers in our model. It is possible that the outsourcing companies receive sensitive data to work on and as the outsourcing companies are not necessarily trusted by the organization, fingerprinting is used on transferred documents. The outsourcing company itself can outsource tasks to other outsourcing companies and thus relay the documents, again using fingerprinting.

If now at any point one of the involved outsourcing companies leaks a confidential document, the organization can invoke the auditor to find the responsible party. The auditor then examines the fingerprints in the document, creates a lineage and is finally able to name the guilty party. In the example given in the introduction [5], there were three outsourcing companies involved and a data leakage could not be clearly associated with one of these. Using LIME the responsible party can be clearly found.

6.2 Online Social Network

The second diagram in Figure 7 shows an online social networking scenario. The users of the network are the owners, as they enter their personal information, post messages, etc. The online social network (OSN) uses all this information as a consumer in this scenario. Third party applications that have access to this information in return for some service act as further consumers in this scenario. The users give their information to the OSN which can relay that information to third party applications using fingerprinting.

In the introduction we gave an example where third party applications leaked private information of Facebook users to advertising companies [3]. Although it was possible to determine which applications leaked data, given some leaked information it is not possible to find the responsible application. Using LIME this link can be found and proven.

7 Discussion

7.1 Composability

LIME also allows us to create a lineage for a document that is published as part of a composed object. Consider the following setting: Owner A and Owner B each own a set of database entries D_A and D_B which they both transfer to Consumer 1, who receives marked versions D_A^1 and D_B^1 . Consumer 1 then creates a composed object $D^1 = D_A^1 || D_B^1$ by concatenation and transfers it further to Consumer 2, who receives a marked version $D^2 = D_A^2 || D_B^2$ and then leaks it. If Owner 1 notices that his set of database entries was published as part of the composed object $D^2 = D_A^2 || D_B^2$, he invokes the auditor and provides him with D^2 , D_A^2 , D_A , the watermark and the necessary detection keys. The auditor detects the watermarks in D_A^2 and verifies that this version was transferred to Consumer 1, who proves that he transferred the composed data to Consumer 2 by showing the watermark and the detection keys for the composed document. As Consumer 2 cannot disprove his guilt, he can be held accountable for publishing a version of D_A even though he published it as part of composed data.

7.2 Collusion Resistance

The collusion resistance of our scheme depends on the collusion resistance of the underlying watermarking scheme. Assume several consumers are working together in order to create an untraceable version of a document. Then their approach is to merge the versions they rightfully obtained to create a new version where the watermarks cannot be detected. As the detection of σ is just a detection of a watermark in the complete document, we obviously have the same collusion resistance as the watermarking scheme for this case. The case of the detection of a bit b_i in a part D_i is again just a detection of a watermark, so the collusion resistance is again the same as for the watermarking scheme. However, we have to know which detected bit belongs to which consumer, so that we can still guarantee that the sender cannot frame the receiving consumers. Linking the detected bits to the responsible consumers is possible, as for each consumer a different embedding key was used. As for each part multiple bits might be detectable, the probability for a sender to successfully frame the receiving consumers is less than or equal to the probability of framing a single recipient successfully, as he still would have to guess all the bits correctly. However, we have to note that in order to successfully mount a collusion attack against our scheme, it is sufficient to mount a collusion attack against 1 of the $n + 1$ watermarks that are used, where n is the number of parts the document was split into.

We can conclude that our scheme tolerates collusions to a certain extent, when it is used with a collusion resistant watermark, without losing its key properties.

7.3 Error Tolerance

Depending on the quality of the underlying watermarking scheme, it may be too strong to require that all bits b_i are detected correctly. Therefore, it could be a good idea to introduce some error tolerance. However, we have to keep in mind that this will increase the probability of the sender successfully framing an innocent recipient. There are two different kinds of errors that can occur: the first one is that no bit can be detected, and the second one is that a wrong bit is detected. Assume the document is split into n parts. Tolerating a non-detectable bit increases the probability of successful framing by a factor of 2. Instead of guessing a bitstring $\bar{b} \in \{0, 1\}^n$, it is sufficient to guess $\bar{b} \in \{0, 1\}^{n-1}$. Tolerating a wrong bit is worse, as it increases this probability by a factor of $(n + 1)$. Instead of accepting just the correct bitstring, we also accept all bitstrings that are changed at exactly one position. As there are n positions, we additionally accept n bitstrings; hence the number of accepted bitstrings and thus the probability of guessing one of these is higher by a factor of $n + 1$. If we want to allow some error tolerance while keeping the probability of

successful framing to be small, we have to choose a larger n ; e.g., to tolerate 128 non-detectable bits, we choose $n = 256$ and have the same framing probability as with $n = 128$ and no tolerance.

7.4 Possible Data Distortion

In our experiment, we used a simple splitting algorithm: We split the image into n equally sized squares. However, when we used a strong watermark for the small parts (that is the α -factor used by the Cox algorithm is 0.5), differences between adjacent parts became visible even though the single watermarks are imperceptible. The resulting image can be seen in Figure 6(b) in the appendix. In some cases, this distortion might affect the usability of the document. We stress however, that we were still able to obtain good results with our approach. In Figure 6(a) we used the Cox algorithm with an alpha factor of 0.1 and no distortion is visible.

It might be interesting to investigate if this problem can be circumvented by using more elaborate splitting algorithms. As most watermarking schemes make use of the contiguity of information in the document, this is not a trivial task.

8 Related Work

8.1 Other Models

The model introduced in [35] intends to help the data distributor to identify the malicious agent which leaked the information. In addition, they argue that current watermarking techniques are not practical, as they may embed extra information which could affect agents' work and their level of robustness may be inadequate. In LIME the relationship of data distributor and agents corresponds to the relationship between data owner and consumer and the model could be used as an alternative method to trace the information given to the consumers.

Controlled data disclosure is a well-studied problem in the security literature, where it is addressed using access control mechanisms. Although these mechanisms can control release of confidential information and also prevent accidental or malicious destruction of information, they do not cover propagation of information by a recipient that is supposed to keep the information private. For example, once an individual allows a third party app to access her information from a social network, she can no longer control how that app may redistribute the information. Therefore, the prevalent access control mechanisms are not adequate to resolve the problem of information leakages. Data usage control enforcement systems [41, 29] employ preventive measures to ensure that data is transferred in distributed systems in a controlled manner preserving the well-defined policies. Techniques have been developed for securely distributing data by forming coalitions among the data owners [45]. In controlled environments, such techniques can be composed with our protocols to improve data privacy.

In [47] the authors present the problem of an insider attack, where the data generator consists of multiple single entities and one of these publishes a version of the document. Usually methods for proof-of-ownership or fingerprinting are only applied after completion of the generating process, so all entities involved in the generation process have access to the original document and could possibly publish it without giving credit to the other authors, or also leak the document without being tracked. As presented in the paper, this problem can be solved by the usage of watermarking and possibly even by using complete fingerprinting protocols during the generating phase of the document.

8.2 Other Fingerprinting Protocols

In [39] Poh addresses the problem of accountable data transfer with untrusted senders using the term *fair content tracing*. He presents a general framework to compare different approaches and splits protocols into four categories depending on their utilization of trusted third parties, i.e., no trusted third parties, offline trusted third parties, online trusted third parties and trusted hardware. Furthermore, he introduces the additional properties of recipient anonymity and fairness in association with payment. All presented schemes use watermarking to trace the guilty party and most presented protocols make use of *watermarking in the encrypted domain*, where encrypted watermarks are embedded in encrypted documents [37]. A new scheme presented is based on chameleon encryption [12]. In [43] Sadeghi also examines several fingerprinting schemes and presents new constructions for symmetric, asymmetric and anonymous fingerprinting schemes. The asymmetric scheme uses a homomorphic commitment scheme to compute the fingerprinted version of the document.

Domingo-Ferrer presents the first fingerprinting protocol that makes use of oblivious transfer in [21]. In the scheme, documents are split into smaller parts and for each part two different versions are created. Then the recipient receives one version of each part via oblivious transfer and in return sends a commitment on the received part. The recipient can now be identified by the unique combinations of versions he received. The protocol has several flaws, as discussed in [42] and [16]. The main problem is that a malicious sender can offer the same version twice in the oblivious transfer, so that he will know which version the recipient receives.

Sadeghi [42] and Hanaoka et al. [16] propose different solutions; the former lets the sender open some pairs to validate that they are not equal and the latter uses oblivious transfer with a two-lock cryptosystem where the recipient can compare both versions in encrypted form. However, both proposed solutions have some flaws themselves. The problem is that it is possible to create two different versions with the same watermark, so even if the equality test fails, the two offered versions can still have the same watermark and the sender will know which watermark the recipient received. Also, the fix proposed in [16] ruins the negligible probability of failure, as it does not split the document into parts, but creates n different versions and sends them via 1-out-of- n oblivious transfer.

Domingo-Ferrer presents another protocol based on oblivious transfer in [22], but again the sender can cheat during oblivious transfer. [26] presents another protocol using oblivious transfer. The protocol uses an approach similar to the chameleon encryption [12], and using 1-out-of- n oblivious transfer a decryption key is transmitted so that the sender does not know it. The protocol suffers from the same problems as the one presented in [16]; namely, the sender can guess the key used by the recipient with non-negligible probability $\frac{1}{n}$ and the sender can even cheat in the oblivious transfer by offering the same key n times, so that he will know the key used by the recipient.

We see that all asymmetric fingerprinting protocols based on oblivious transfer that have been proposed so far suffer from the same weakness. We circumvent this problem in our protocol by additionally sending a signed message including the watermark's content, so that the recipient is able to prove what he asked for. In contrast to the watermark, this message can be read by the recipient, so he can notice if the sender cheats.

8.3 Broadcasting

Parviainen and Parnes present an approach for distributing data in a multicast system, so that every recipient holds a differently watermarked version [36]. The sender splits the file into blocks and for each block he creates two different versions by watermarking them with different watermarks and encrypting them with different keys. Each recipient is assigned a set of keys, so that he can decrypt exactly one version of each part. The resulting combination of parts can uniquely identify the recipient. In [9] Adelsbach, Huber and Sadeghi show another approach for a broadcasting system that allows identification of recipients by their

received files. With a technique called *fingercasting*, recipients automatically embed a watermark in files during the decryption process. The process is based on the chameleon cipher [12], which allows one to decrypt an encrypted file with different decryption keys, to introduce some noise that can be used as a means of identification. In [28] Katzenbeisser et al. use the technique of fingercasting together with a randomized fingerprinting code in order to provide better security against colluding attackers. However, in these broadcasting approaches the problem of an untrusted sender is not addressed.

8.4 Watermarking

LIME can be used with any type of data for which watermarking schemes exist. Therefore, we briefly describe different watermarking techniques for different data types. Most watermarking schemes are designed for multimedia files such as images [40], videos [20], and audio files [11]. In these multimedia files, watermarks are usually embedded by using a transformed representation (e.g. discrete cosine, wavelet or Fourier transform) and modifying transform domain coefficients.

Watermarking techniques have also been developed for other data types such as relational databases, text files and even Android apps. The first two are especially interesting, as they allow us to apply LIME to user databases or medical records. Watermarking relational databases can be done in different ways. The most common solutions are to embed information in noise-tolerant attributes of the entries or to create fake database entries [25]. For watermarking of texts, there are two main approaches. The first one embeds information by changing the text’s appearance (e.g. changing distance between words and lines) in a way that is imperceptible to humans [15]. The second approach is also referred to as language watermarking and works on the semantic level of the text rather than on its appearance [13]. A Mechanism also has been proposed to insert watermarks to Android apps [49]. This mechanism encodes a watermark in a permutation graph and hides the graph as a linked list in the application. Due to the list representation, watermarks are encoded in the execution state of the application rather than in its syntax, which makes it robust against attacks.

Embedding multiple watermarks into a single document has been discussed in literature and there are different techniques available [46, 32, 23]. In [32] they discuss multiple re-watermarking and in [23] the focus is on segmented watermarking. Both papers show in experimental results that multiple watermarking is possible which is very important for our scheme, as it allows us to create a lineage over multiple levels.

It would be desirable not to reveal the private watermarking key to the auditor during the auditor’s investigation, so that it can be safely reused, but as discussed in [18, 31, 48] current public key watermarking schemes are not secure and it is doubtful if it is possible to design one that is secure. In [44] Sadeghi presents approaches to zero-knowledge watermark detection. With this technology it is possible to convince another party of the presence of a watermark in a document without giving any information about the detection key or the watermark itself. However, the scheme discussed in [44] also hides the content of the watermark itself and are therefore unfit for our case, as the auditor has to know the watermark to identify the guilty person. Furthermore, using a technology like this would come with additional constraints for the chosen watermarking scheme.

9 Conclusion and Future Directions

We present LIME, a model for accountable data transfer across multiple entities. We define participating parties, their inter-relationships and give a concrete instantiation for a data transfer protocol using a novel combination of oblivious transfer, robust watermarking and digital signatures. We prove its correctness and show that it is realizable by giving microbenchmarking results. By presenting a general applicable framework, we introduce accountability as early as in the design phase of a data transfer infrastructure.

Although LIME does not actively prevent data leakage, it introduces reactive accountability. Thus, it will deter malicious parties from leaking private documents and will encourage honest (but careless) parties to provide the required protection for sensitive data. LIME is flexible as we differentiate between trusted senders (usually owners) and untrusted senders (usually consumers). In the case of the trusted sender, a very simple protocol with little overhead is possible. The untrusted sender requires a more complicated protocol, but the results are not based on trust assumptions and therefore they should be able to convince a neutral entity (e.g. a judge).

Our work also motivates further research on data leakage detection techniques for various document types and scenarios. For example, it will be an interesting future research direction to design a verifiable lineage protocol for derived data.

Acknowledgements. We thank Deepak Garg for encouraging discussions and Amit Roy, Emre Goynugur and Isaac Alpizar Chacon for their work towards an initial draft. We are also grateful to our anonymous reviewers for their comments.

References

- [1] Chronology of data breaches. <http://www.privacyrights.org/data-breach>.
- [2] Data breach cost. http://www.symantec.com/about/news/release/article.jsp?prid=20110308_01.
- [3] Facebook in Privacy Breach. <http://online.wsj.com/article/SB10001424052702304772804575558484075236968.html>.
- [4] GNU Multiple Precision Arithmetic Library (GMP). <http://gmplib.org/>.
- [5] Offshore outsourcing. http://www.computerworld.com/s/article/109938/Offshore_outsourcing_cited_in_Florida_data_leak.
- [6] Pairing-Based Cryptography Library (PBC). <http://crypto.stanford.edu/pbc>.
- [7] Privacy rights clearinghouse. <http://www.privacyrights.org>.
- [8] Electronic Privacy Information Center (EPIC). <http://epic.org>, 1994.
- [9] A. Adelsbach, U. Huber, and A.-R. Sadeghi. Fingercasting - Joint Fingerprinting and Decryption of Broadcast Messages. *T. Data Hiding and Multimedia Security*, 2:1–34, 2007.
- [10] A. Adelsbach, S. Katzenbeisser, and A.-R. Sadeghi. A computational model for watermark robustness. In *Information Hiding*, pages 145–160. Springer, 2007.
- [11] M. A. Alsalamy and M. M. Al-Akaidi. Digital audio watermarking: survey. *School of Engineering and Technology, De Montfort University, UK*, 2003.
- [12] R. Anderson and C. Manifavas. Chameleon - A new kind of stream cipher. In *Fast Software Encryption*, pages 107–113. Springer, 1997.
- [13] M. J. Atallah, V. Raskin, C. Hempelmann, M. Karahan, R. Sion, U. Topkara, and K. E. Triezenberg. Natural Language Watermarking and Tamperproofing. In *Information Hiding*, pages 196–212, 2002.
- [14] D. Boneh, B. Lynn, and H. Shacham. Short signatures from the Weil pairing. In *Advances in Cryptology-ASIACRYPT 2001*, pages 514–532. Springer, 2001.
- [15] J. T. Brassil, S. Low, and N. F. Maxemchuk. Copyright protection for the electronic distribution of text documents. *Proceedings of the IEEE*, 87(7):1181–1196, 1999.
- [16] J.-G. Choi, G. Hanaoka, K. H. Rhee, and H. Imai. How to break COT-based fingerprinting schemes and design new one. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 88(10):2800–2807, 2005.
- [17] I. J. Cox, J. Kilian, F. T. Leighton, and T. Shamoon. Secure spread spectrum watermarking for multimedia. *Image Processing, IEEE Transactions on*, 6(12):1673–1687, 1997.
- [18] I. J. Cox and J.-P. M. Linnartz. Public watermarks and resistance to tampering. In *International Conference on Image Processing (ICIP'97)*, pages 26–29, 1997.
- [19] W. Dai. Crypto++ Library. <http://cryptopp.com>.

- [20] G. Doërr and J.-L. Dugelay. A guide tour of video watermarking. *Signal processing: Image communication*, 18(4):263–282, 2003.
- [21] J. Domingo-Ferrer. Anonymous fingerprinting based on committed oblivious transfer. In *Public Key Cryptography*, pages 43–52. Springer, 1999.
- [22] J. Domingo-Ferrer and J. Herrera-Joancomartí. Efficient smart-card based anonymous fingerprinting. In *Smart Card Research and Applications*, pages 221–228. Springer, 2000.
- [23] K. Frikken and M. Atallah. Cropping-resilient segmented multiple watermarking. In *Algorithms and Data Structures*, pages 231–242. Springer, 2003.
- [24] S. Goldwasser, S. Micali, and R. L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Comput.*, 17(2):281–308, 1988.
- [25] R. Halder, S. Pal, and A. Cortesi. Watermarking techniques for relational databases: Survey, classification and comparison. *Journal of Universal Computer Science*, 16(21):3164–3190, 2010.
- [26] D. Hu and Q. Li. Asymmetric fingerprinting based on 1-out-of-n oblivious transfer. *Communications Letters, IEEE*, 14(5):453–455, 2010.
- [27] Y. Ishai, J. Kilian, K. Nissim, and E. Petrank. Extending oblivious transfers efficiently. In *Advances in Cryptology-CRYPTO 2003*, pages 145–161. Springer, 2003.
- [28] S. Katzenbeisser, B. Skoric, M. U. Celik, and A.-R. Sadeghi. Combining Tardos Fingerprinting Codes and Fingercasting. In *Information Hiding*, pages 294–310, 2007.
- [29] F. Kelbert and A. Pretschner. Data usage control enforcement in distributed systems. In *CODASPY*, pages 71–82, 2013.
- [30] J. Kilian, F. T. Leighton, L. R. Matheson, T. G. Shamoan, R. E. Tarjan, and F. Zane. Resistance of digital watermarks to collusive attacks. In *IEEE International Symposium on Information Theory*, pages 271–271, 1998.
- [31] J.-P. M. Linnartz and M. Van Dijk. Analysis of the sensitivity attack against electronic watermarks in images. In *Information Hiding*, pages 258–272. Springer, 1998.
- [32] A. Mascher-Kampfer, H. Stögner, and A. Uhl. Multiple re-watermarking scenarios. In *Proceedings of the 13th International Conference on Systems, Signals, and Image Processing (IWSSIP 2006)*, pages 53–56. Citeseer, 2006.
- [33] P. Meerwald. Watermarking toolbox. <http://www.cosy.sbg.ac.at/~pmeerw/Watermarking/source>.
- [34] M. Naor and B. Pinkas. Efficient oblivious transfer protocols. In *Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 448–457, 2001.
- [35] P. Papadimitriou and H. Garcia-Molina. Data leakage detection. *Knowledge and Data Engineering, IEEE Transactions on*, 23(1):51–63, 2011.
- [36] R. Parviainen and P. Parnes. Large scale distributed watermarking of multicast media through encryption. In *Proceedings of the IFIP TC6/TC11 International Conference on Communications and Multimedia Security Issues of the New Century*, volume 192, pages 149–158, 2001.
- [37] R. Petrovic and B. Tehranchi. Watermarking in an encrypted domain, July 7 2006. US Patent App. 11/482,519.
- [38] B. Pfitzmann and M. Waidner. Asymmetric fingerprinting for larger collusions. In *Proceedings of the 4th ACM conference on Computer and communications security, CCS '97*, pages 151–160, 1997.
- [39] G. S. Poh. *Design and Analysis of Fair Content Tracing Protocols*. PhD thesis, 2009.
- [40] V. M. Potdar, S. Han, and E. Chang. A survey of digital image watermarking techniques. In *International Conference on Industrial Informatics, 2005. INDIN'05. 2005 3rd IEEE*, pages 709–716. IEEE, 2005.
- [41] A. Pretschner, M. Hilty, F. Schütz, C. Schaefer, and T. Walter. Usage Control Enforcement: Present and Future. *IEEE Security & Privacy*, 6(4):44–53, 2008.
- [42] A.-R. Sadeghi. How to break a semi-anonymous fingerprinting scheme. In *Information Hiding*, pages 384–394. Springer, 2001.
- [43] A.-R. Sadeghi. *Secure Fingerprinting on Sound Foundations*. PhD thesis, 2004.
- [44] A.-R. Sadeghi. The Marriage of Cryptography and Watermarking – Beneficial and Challenging for Secure Watermarking and Detection. In *Proceedings of the 6th International Workshop on Digital Watermarking, IWDW '07*, pages 2–18, 2008.
- [45] F. Salim, N. P. Sheppard, and R. Safavi-Naini. A Rights Management Approach to Securing Data Distribution in Coalitions. In *NSS*, pages 560–567, 2010.
- [46] N. P. Sheppard, R. Safavi-Naini, and P. Ogunbona. On multiple watermarking. In *MM&Sec*, pages 3–6, 2001.

- [47] N. P. Sheppard, R. Safavi-Naini, and P. Ogunbona. Secure multimedia authoring with dishonest collaborators. *EURASIP J. Appl. Signal Process.*, 2004:2214–2223, 2004.
- [48] Y. Wu and R. H. Deng. A Pollution Attack to Public-key Watermarking Schemes. In *Multimedia and Expo (ICME), 2012 IEEE International Conference on*, pages 230–235. IEEE, 2012.
- [49] W. Zhou, X. Zhang, and X. Jiang. Appink: watermarking android apps for repackaging deterrence. In *Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security*, pages 1–12. ACM, 2013.